CPE301 – SPRING 2022

Design Assignment 3

Student Name: Ernesto Ibarra-Ayala
Student #: 2001211571
Student Email: ibarre3@unlv.nevada.edu
Primary Github address: https://github.com/Ernesto-Ibarra/Work.git

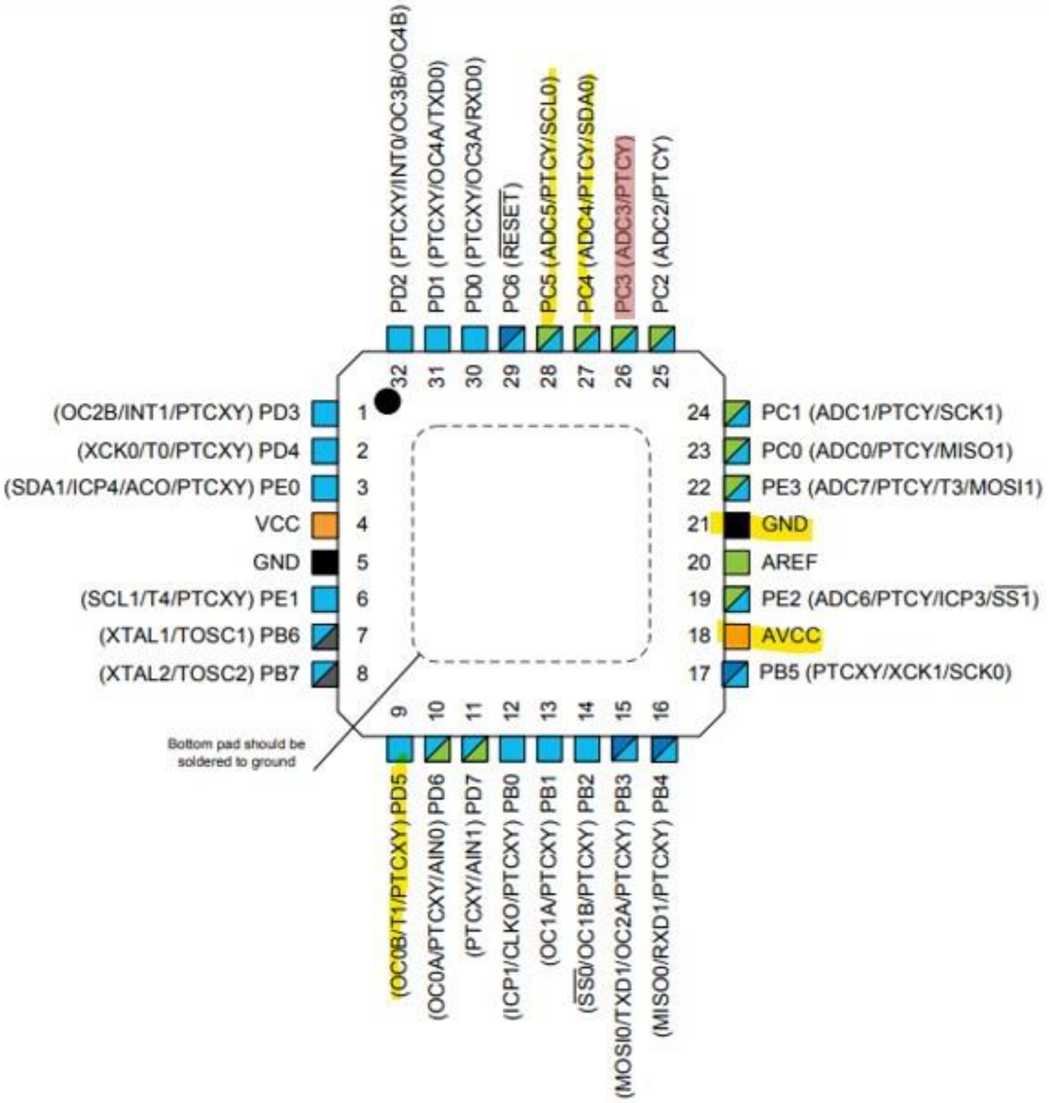
1. COMPONENTS LIST AND CONNECTION BLOCK DIAGRAM w/ PINS

Atmel Studio 7.0
- Assembler
- Simulator
- Debugger

Atmega328PB-Xmini

Multi-Function Shield
- Switches
- LEDs

Logic Analyzer



2. INITIAL/MODIFIED/DEVELOPED CODE OF TASK 1/2/3

```
/*
         * ADC2_example.c
         *
         * Created: 10/10/2019 10:24:06 PM
         * Author : VenkatesanMuthukumar
         */

        #include <avr/io.h>
        #include <stdlib.h>
        #define F_CPU 16000000UL
        #include <util/delay.h>
        #define BAUDRATE 9600
        #define BAUD_PRESCALLER (((F_CPU / (BAUDRATE * 16UL))) - 1)

        uint16_t adc_value;            //Variable used to store the value read from the ADC
        char buffer[5];                //Output of the itoa function
        uint8_t i=0;                //Variable for the for() loop

        void adc_init(void);            //Function to initialize/configure the ADC
        uint16_t read_adc(uint8_t channel);    //Function to read an arbitrary analogic
        channel/pin
        void USART_init(void);            //Function to initialize and configure the
        USART/serial
        void USART_send( unsigned char data);    //Function that sends a char over the
        serial port
        void USART_putstring(char* StringPtr);    //Function that sends a string over the
        serial port

        int main(void){
               adc_init();        //Setup the ADC
               USART_init();        //Setup the USART

               for(;;){        //Our infinite loop
               /*     for(i=0; i<3; i++){
                             //USART_putstring("Reading channel ");
                             //USART_send('0');            //This is a nifty trick when we
        only want to send a number between 0 and 9
                             //USART_putstring(" : ");            //Just to keep things
        pretty
                             adc_value = read_adc(i);        //Read one ADC channel
```

```
                      itoa(adc_value, buffer, 10);        //Convert the read value
to an ascii string
                      USART_putstring(buffer);        //Send the converted value to
the terminal
                      USART_putstring(",");            //Some more formatting
                      _delay_ms(500);                //You can tweak this value to
have slower or faster readings or for max speed remove this line
               */
               adc_value = read_adc(4);        //Read one ADC channel
               itoa(adc_value, buffer, 10);        //Convert the read value to an
ascii string
               USART_putstring(buffer);        //Send the converted value to the
terminal
               USART_putstring(",");            //Some more formatting
               adc_value = read_adc(5);        //Read one ADC channel
               itoa(adc_value, buffer, 10);        //Convert the read value to an
ascii string
               USART_putstring(buffer);        //Send the converted value to the
terminal
               USART_send('\n');             //Some more formatting
               //}
               //USART_send('\r');
               //USART_send('\n');                //This two lines are to tell to
the terminal to change line
       */

       }

       return 0;
}

void adc_init(void){
       ADCSRA |= ((1<<ADPS2)|(1<<ADPS1)|(1<<ADPS0));    //16Mhz/128 = 125Khz the
ADC reference clock
       ADMUX |= (1<<REFS0);                //Voltage reference from Avcc (5v)
       ADCSRA |= (1<<ADEN);                //Turn on ADC
       ADCSRA |= (1<<ADSC);                //Do an initial conversion because this
one is the slowest and to ensure that everything is up and running
}

uint16_t read_adc(uint8_t channel){
       ADMUX &= 0xF0;                    //Clear the older channel that was read
       ADMUX |= channel;                //Defines the new ADC channel to be read
```

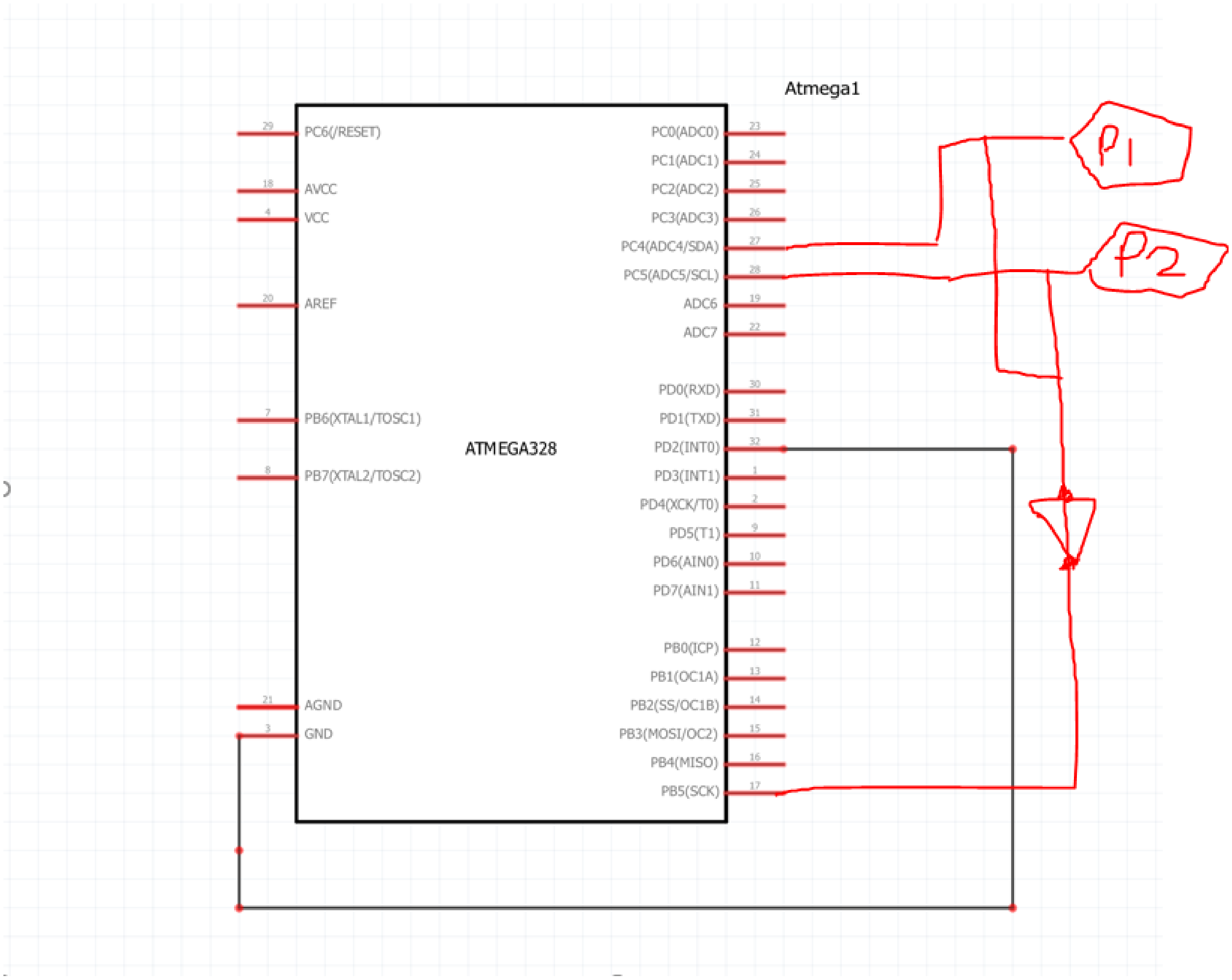
```
        ADCSRA |= (1<<ADSC);                //Starts a new conversion
        while(ADCSRA & (1<<ADSC));            //Wait until the conversion is done
        return ADCW;                    //Returns the ADC value of the chosen 
channel
}

void USART_init(void){

        UBRR0H = (uint8_t)(BAUD_PRESCALLER>>8);
        UBRR0L = (uint8_t)(BAUD_PRESCALLER);
        UCSR0B = (1<<RXEN0)|(1<<TXEN0);
        UCSR0C = (3<<UCSZ00);
}

void USART_send( unsigned char data){

        while(!(UCSR0A & (1<<UDRE0)));
        UDR0 = data;

}

void USART_putstring(char* StringPtr){

        while(*StringPtr != 0x00){
                USART_send(*StringPtr);
        StringPtr++;}

}
```

3. DEVELOPED MODIFIED CODE OF TASK 1/2/3

```
#define F_CPU 16000000UL //Set clock frequency
#define BAUDRATE 9600 //Set the baud rate
#define BAUD_PRESCALLER (((F_CPU / (BAUDRATE * 16UL))) - 1) //Prescalar for the baud rate
#include <avr/io.h>
#include <stdlib.h>
#include <util/delay.h>
#include <avr/interrupt.h>


uint16_t ADCX; //ADC X-axis
uint16_t ADCY; //ADC Y-axis
char buffer[5]; //String for output of string function
float XF = 0.0;      //x-axis value to frequency for timer
float YDC = 0.0; //y-axis value to duty cycle for timer
float percent = 0.0;// Stores value of duty cycle to convert to a percentage
float Hzz = 0.0;// variable used to display frequency

void adc_init(void); //Function to initialize/configure the ADC
uint16_t read_adc(uint8_t channel); //Function to read the analog input
void USART_init(void); //Function to initialize and configure the USART/serial
void USART_send( unsigned char data); //Function that sends a char over the serial port
void USART_putstring(char* StringPtr); //Function that sends a string over the serial
port
void adc_convert(void); //Main function used to read the ADC values and use our timbers
with those values
void timer_init(void); //Function to initialize the CTC timer1
void timer_update(void); //Function to update the timer values when new ADC values come
in

int main(void){
        DDRC &= (0 << 4) | (0 << 5); //Set PC4 and PC5 as input
        DDRB |= (1 << 5); //set PB5 as output
        PORTB &= (0 << 5);
        adc_init(); //Start ADC
        USART_init(); //Start the USART
        timer_init(); //Start Timer1

        while(1)
        {
                adc_convert(); //Grab ADC values and process them
                timer_update(); //Update timer
        }

        return 0;
}


void adc_convert(void)
{
        // Here is part 2 of the HW here we will display the RAW values of the ADC
        ADCX = read_adc(4); //Read ADC channel 4
        USART_putstring("RAW FREQ: ");
        itoa(ADCX, buffer, 10);
        USART_putstring(buffer);
        USART_putstring(",");
        ADCY = read_adc(5); //Read ADC channel 5
        USART_putstring("RAW DUTY: ");
        itoa(ADCY, buffer, 10);
        USART_putstring(buffer);
        USART_send('\n');

        //Here we take care of part 3 of the assignment by using this IF statements to
create the correct Frequency and Duty Cycle
        if (ADCX >= 512)
        {
                XF = 260 + (1023-ADCX)/2; //If valX > 512 it will use this formula to scale
down
        }
        else
        {
                XF = 260+(512-ADCX)*15; //If valX < 512 it will use this formula to scale
up
        }

        YDC = XF * (1.0 * ADCY / 1000.0); //This formula will convert valY into a
percentage to get the duty cycle

        if(YDC >= XF)
        {
                YDC = XF;
        }

        percent = ADCY / 10; //This will give us the duty cycle from 0 - 100
        //Hzz = XF
        USART_putstring("Freq: ");
        itoa(XF, buffer, 10);
        USART_putstring(buffer);
        USART_putstring(",");
        USART_putstring("DUTY: ");
        itoa(percent, buffer, 10);
        USART_putstring(buffer);
        USART_putstring("%");
        USART_send('\n');
        _delay_ms(500);
}

void adc_init(void)
{
        ADCSRA |= ((1<<ADPS2)|(1<<ADPS1)|(1<<ADPS0));    //16Mhz/128 = 125Khz the ADC
reference clock
        ADMUX |= (1<<REFS0);                //Voltage reference from Avcc (5v)
        ADCSRA |= (1<<ADEN);                //Turn on ADC
        ADCSRA |= (1<<ADSC);                //Do an initial conversion because this one is
the slowest and to ensure that everything is up and running
}

uint16_t read_adc(uint8_t channel)
{
        ADMUX &= 0xF0;                     //Clear the older channel that was read
        ADMUX |= channel;                 //Defines the new ADC channel to be read
        ADCSRA |= (1<<ADSC);                //Starts a new conversion
        while(ADCSRA & (1<<ADSC));            //Wait until the conversion is done
        return ADCW;                     //Returns the ADC value of the chosen channel
}

void timer_init(void)
{
        TCCR1B |= (1 << WGM12) | (1 << CS12) | (1 << CS10); //Sets prescalar to 1024
        TIMSK1 |= (1 << OCIE1A) | (1 << OCIE1B); //Enable OCR1A and OCR1B
        TCNT1 = 0; // Start timer at 0
        sei(); //Activate global interrupts
}

void timer_update(void)
{
        OCR1A = XF; //Store new frequency value
        OCR1B = YDC; //Store new duty cycle value
}
```

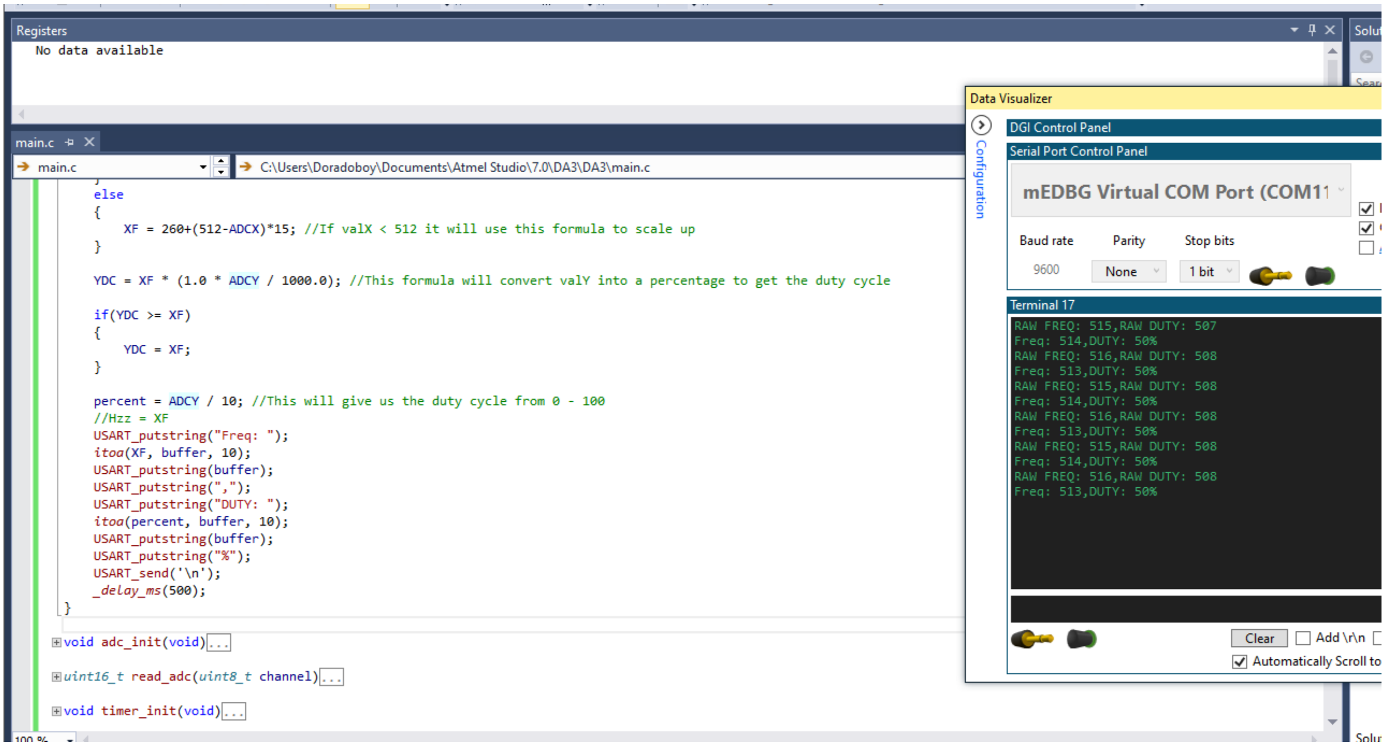
```
ISR(TIMER1_COMPB_vect) //Interrupt for duty cycle
{
	PORTB |= (1 << 5); //turn on the LED
}

ISR(TIMER1_COMPA_vect)  //Interrupt for frequency
{
	PORTB &= (0 << 5); // turn off LED
	TCNT1 = 0; //Reset to zero
}

void USART_init(void)//Function to initialize the USART
{

	UBRR0H = (uint8_t)(BAUD_PRESCALLER>>8);
	UBRR0L = (uint8_t)(BAUD_PRESCALLER);
	UCSR0B = (1<<RXEN0)|(1<<TXEN0);
	UCSR0C = (3<<UCSZ00);
}

void USART_send( unsigned char data)//Function to send data using USART
{

	while(!(UCSR0A & (1<<UDRE0)));
	UDR0 = data;

}

void USART_putstring(char* StringPtr)//Function to turn send a string on USART
{

	while(*StringPtr != 0x00){
		USART_send(*StringPtr);
	StringPtr++;}

}
```

4. SCHEMATICS



5. SCREENSHOTS OF EACH TASK OUTPUT (ATMEL STUDIO OUTPUT)

Here is an output of my code which will take care of all three parts of the assignment. Below we can see the Raw ADC Values right above the frequency and duty cycle values. I added a small delay of 500ms to allow me to be able to screenshot the results.



```
    else
    {
        XF = 260+(512-ADCX)*15; //If valX < 512 it will use this formula to scale up
    }

    YDC = XF * (1.0 * ADCY / 1000.0); //This formula will convert valY into a percentage to get the duty cycle

    if(YDC >= XF)
    {
        YDC = XF;
    }

    percent = ADCY / 10; //This will give us the duty cycle from 0 - 100
    //Hzz = XF
    USART_putstring("Freq: ");
    itoa(XF, buffer, 10);
    USART_putstring(buffer);
    USART_putstring(",");
    USART_putstring("DUTY: ");
    itoa(percent, buffer, 10);
    USART_putstring(buffer);
    USART_putstring("%");
    USART_send('\n');
    _delay_ms(500);
}

void adc_init(void)...

uint16_t read_adc(uint8_t channel)...

void timer_init(void)...
```

mEDBG Virtual COM Port (COM11

Baud rate: 9600 Parity: None Stop bits: 1 bit

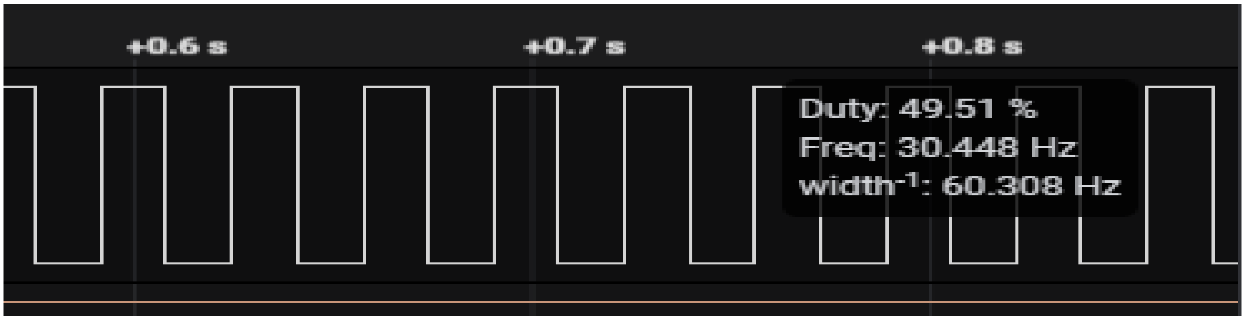
☑
☑
□

```
RAW FREQ: 515,RAW DUTY: 507
Freq: 514,DUTY: 50%
RAW FREQ: 516,RAW DUTY: 508
Freq: 513,DUTY: 50%
RAW FREQ: 515,RAW DUTY: 508
Freq: 514,DUTY: 50%
RAW FREQ: 516,RAW DUTY: 508
Freq: 513,DUTY: 50%
RAW FREQ: 515,RAW DUTY: 508
Freq: 514,DUTY: 50%
RAW FREQ: 516,RAW DUTY: 508
Freq: 513,DUTY: 50%
```

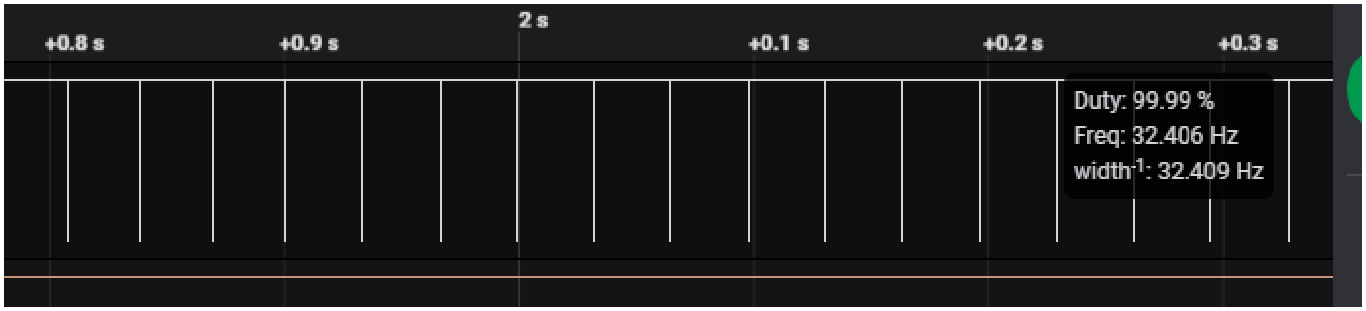
□ Add \r\n □

☑ Automatically Scroll to

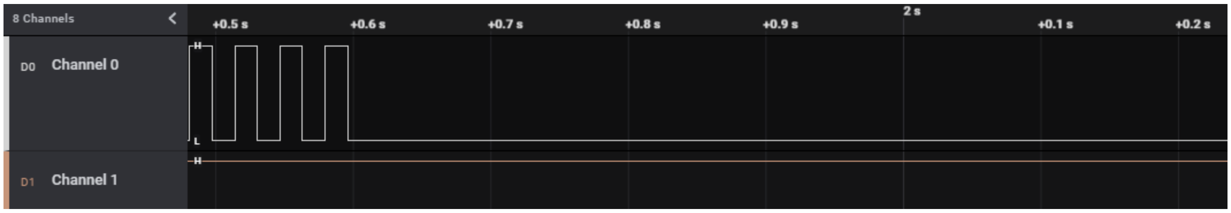
Here is me using the data analyzer to get the correct values for the Joystick. This is joystick being in the center so we can see the Duty cycle is 50% and the Frequency is 30Hz which is what we wanted to display.



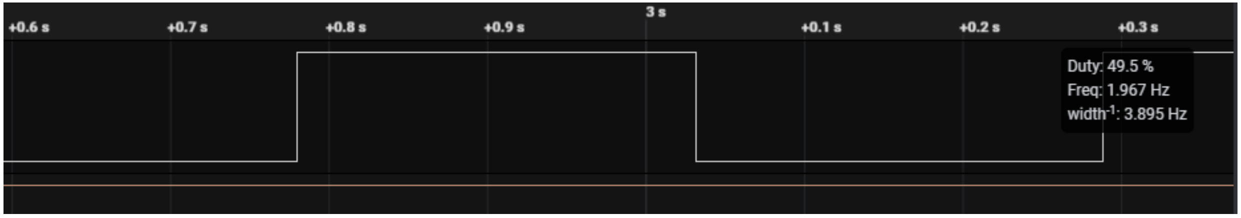
Here we can see what happens when we move the Joystick all the way to one extreme, we get the duty cycle being 100% while the frequency remains unchanged at 30Hz.



Here we can see what happens when we move the Joystick to the opposite end, we can basically turn off the LED.

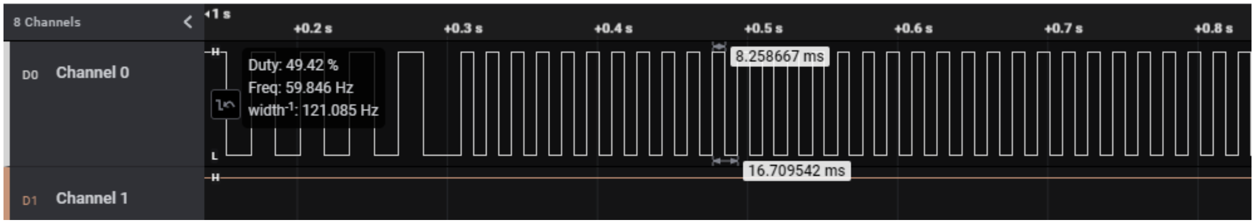


Here is me changing the Frequency to one extreme causing it to go to zero or at least very close to zero which is basically 1Hz while the Duty cycle remains at 50% as you can see below.



Duty: 49.5 %
Freq: 1.967 Hz
width^{-1}: 3.895 Hz

Here we can see changing the Frequency to the other extreme causing it to have a frequency of 60Hz as we wanted on the assignment, while the Duty cycle remains at 50% as stated. So we are basically able to see all 4 quadrants of the Joystick.

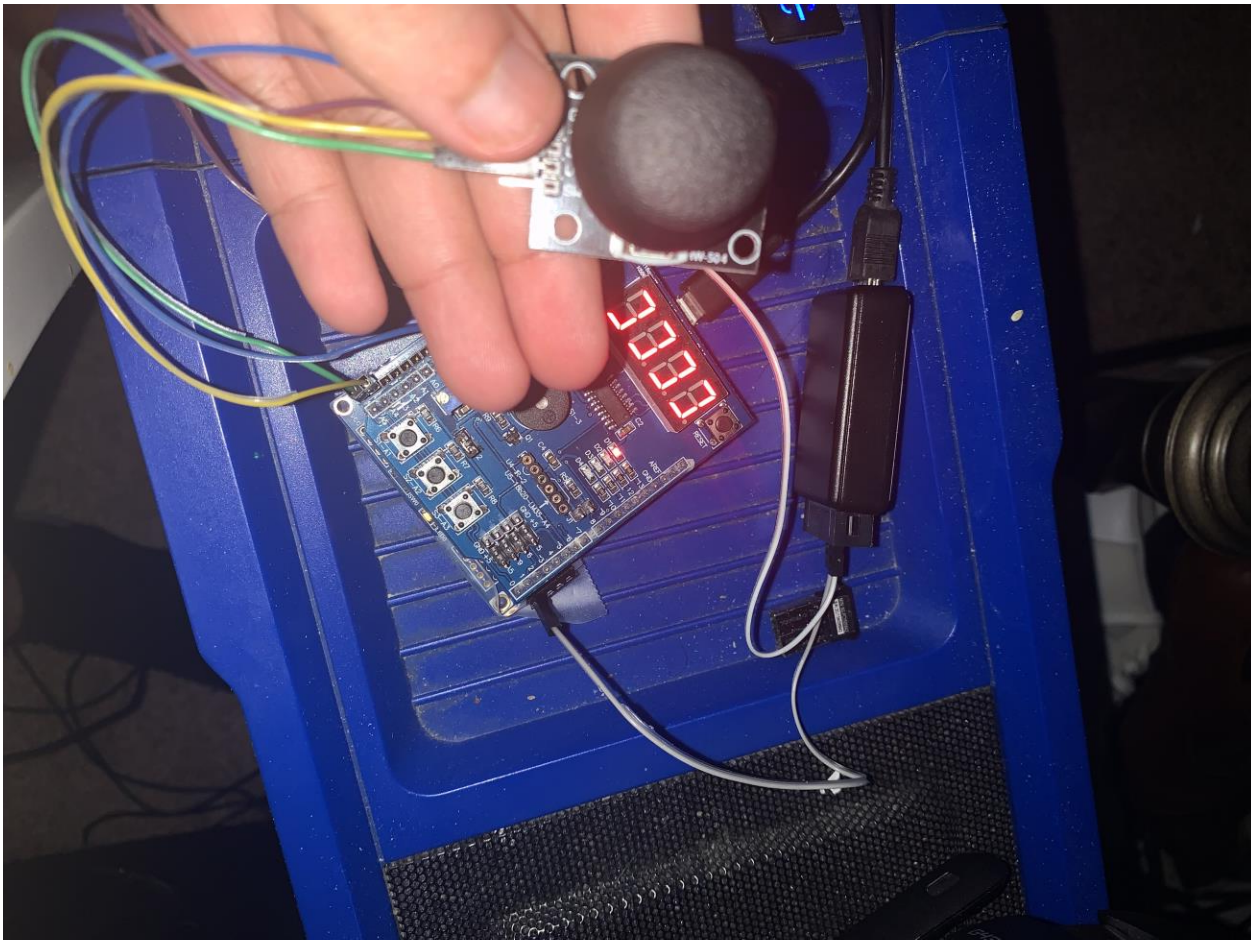


Duty: 49.42 %
Freq: 59.846 Hz
width^{-1}: 121.085 Hz

8.258667 ms

16.709542 ms

6. SCREENSHOT OF EACH DEMO (BOARD SETUP)



7. VIDEO LINKS OF EACH DEMO

DA3 Part 2: https://youtube.com/shorts/j1wdf_MSlik?feature=share

DA3 Part 3: https://youtu.be/NsZ8_ZAMUuU

8. GITHUB LINK OF THIS DA

https://github.com/Ernesto-Ibarra/Work/tree/main/DesignAssignments

Student Academic Misconduct Policy
http://studentconduct.unlv.edu/misconduct/policy.html

"This assignment submission is my own, original work".
Ernesto Ibarra